Application of HLA to Distributed Virtual Ship Combat Information Center Training

Suleyman Guleyupoglu
Patrick Melody
ITT Industries, Inc.
2560 Huntington Avenue
Alexandria, VA 22303
202-767-8023, 202-767-3879
suleyman@ait.nrl.navy.mil, melody@ait.nrl.navy.mil

Henry Ng
U.S. Naval Research Laboratory
4555 Overlook Avenue, SW
Washington, DC 20375
202-767-6023
ng@ait.nrl.navy.mil

Keywords:
HLA, RTI, Ownership Management, CIC, Distributed Simulation, Performance, Virtual Reality, Training.

ABSTRACT: *The High Level Architecture (HLA) and its software implementation Runtime Infrastructure (RTI) are developed by Defense Modeling and Simulation Office (DMSO) to provide a common framework for disparate simulations to communicate with one another. This paper presents lessons learned from making one of the U.S. Navy distributed simulations comply with HLA.*

Virtual ship combat information center, or Virtual CIC, is a virtual reality application developed as a cost-effective means of training crew members at Navy surface combatant training facilities. Real combat system consoles are used for crew training in these facilities. Due to the high cost of acquiring, reconfiguring and maintaining these consoles, there is a limited number of consoles available to the students. With the advent of cheaper and more powerful graphics computers, virtual reality is an economical and viable training alternative. An added benefit of the Virtual CIC is the ability to conduct training in a distributed environment where students and the instructor can be at different geographical locations but interact with each other in the same virtual ship or CIC. Virtual CIC was developed to use User Datagram Protocol (UDP) to communicate between each participant. In order to comply with the Department of Defense *(DoD) mandate, Virtual CIC was upgraded to use RTI.*

RTI Version 1.3R6 provided by DMSO was used in the upgrade of Virtual CIC. RTI ownership management services are utilized to manage access to common resources in order to maintain a consistent tactical picture among crewmembers in a distributed training session. A number of observations were made during the implementation and testing of the system. This paper presents those observations in some detail.

1. Introduction

The High Level Architecture (HLA) for Modeling and Simulation (M&S) is a network communication architecture designed to provide a mechanism for heterogeneous simulations to coordinate their run-time execution [1]. The creation of coordinated executions is designed to save money by allowing specialized resources (e.g. airplane simulators, custom computer hardware, live platform interfaces) to be shared by multiple programs. However, because the HLA has only recently been developed as the DoD Distributed Simulation standard, information on the applicability of this standard to Navy M&S problems should be tested in real-world applications.

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
2010		**00-00-2010 to 00-00-2010**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Application of HLA to Distributed Virtual Ship Combat Information Center Training	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **ITT Industries, Inc,2560 Huntington Avenue,Alexandria,VA,20375**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**	**Same as Report (SAR)**	**10**	

This paper presents performance characteristics of the HLA Run-Time Infrastructure (RTI) in supporting a Navy application. RTI is the backbone of the High Level Architecture, through which information is passed from one system to another. The RTI has components intended to support widely disparate simulation systems.

Simulations *subscribe* to the RTI for specific *services* and likewise announce their intent to *publish* specific types of information. Thus, simulations become *clients* for some types of information or *servers* of information or both. This "client/server paradigm" is a radical departure from the structure of DIS-based simulation systems and will have an impact on the simulations using the paradigm.

It is not clear how the RTI will actually perform in the presence of multiple services contending for available communications bandwidth. Naval system simulations will require dynamic information that is likely to be available only from powerful, dedicated server systems. Examples include environmental services such as volumetric atmospheric and oceanic data, long range or shallow water acoustic computations, electro-optic sensor computations, voice and digital communications network performance, and complex Command and Control and Decision Support systems.

In order to evaluate the performance characteristics of RTI, an existing simulation was upgraded to use RTI. The application is a virtual reality application that simulates a ship Combat Information Center (CIC). In this application the events will have to be communicated to each participant with minimum latency and the tactical picture consistency must be maintained. RTI's ownership management services play a vital role for achieving this goal and should be performed by RTI effectively. Therefore this application was used to evaluate RTI capabilities in supporting real time applications as well as ownership handoff performance. We describe the Virtual CIC application in the next section.

2. Virtual CIC

As the US defense budget decreases, the US Navy training community is more cost conscious than ever. In current US shore-based surface combatant training facilities new crews are trained using real combat consoles in a classroom environment. This has the considerable expense of supporting, maintaining, and reconfiguring the consoles used in these facilities. In addition, there are a limited number of training consoles available, limiting time available to each student to use.

Virtual reality technology has become a cost-effective training solution with the advent of more powerful and cheaper graphics computers. This technology allows the user to be immersed into a simulated graphical environment where he or she sees the virtual environment through a computer monitor or a head mounted display and provides input through various devices such as a keyboard, mouse or more sophisticated alternatives. We coupled virtual reality and distributed simulation technology to develop a distributed virtual ship Combat Information Center (CIC) for the surface ship training. The goal is to create a distributed immersive learning environment to complement current training capability with reduced maintenance cost and higher availability. CIC crewmembers can then perform team learning and training in a virtual ship environment. The virtual environment offers a true interactive 3D view of the interior of the CIC as shown in Figure 2.1. The visual simulation portion of the Virtual CIC is intended to provide the "look and feel" of the actual CIC through the use of extensive 3D models, phototextures, and sound clips. Virtual environments create a new level of training by making the training experience "realistic" instead of a classroom-like environment. It also simulates the equipment that the crew interacts with to perform detection, classification and target engagement activities. Each student appears in the environment as a 3D human representation, an avatar. A picture of each student's face is scanned and mapped onto his avatar with the 3D texture mapping technique so that each student can recognize the others inside the Virtual CIC. The Virtual CIC's networking capability allows students at different geographical locations to train together in a unified virtual environment over a wide area network. This networking capability not only allows the crews in the same ship CIC to be distributed at different locations for training, it also provides the capability to simulate the whole virtual battle group operations (multiple ships) as shown in Figure 2.2.

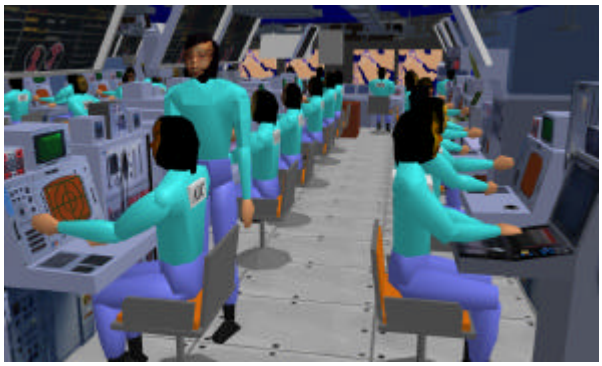

Figure 2.1: Virtual Ship Combat Information Center

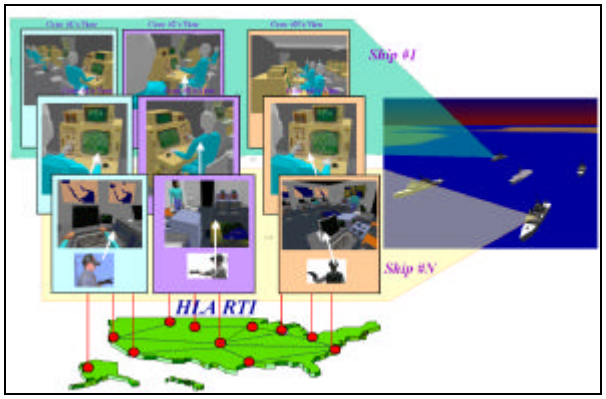

Figure 2.2: Virtual Battle Group

Figure 2.3: Holocube Visualization Aid

Various information visualization aids were incorporated into the virtual environment to help students in learning different tactical deployment of the combat systems. For example, we have created a "holocube" that allows the student to visualize the entire battlespace as shown in Figure 2.3. This allows students to correlate sensor data with a visual 3D display of a god's-eye view, facilitating understanding of sensor capabilities and operations. This holographic-like display could also allow for visualization of sensor coverage, emission restrictions, operational boundaries and other doctrinal concepts and entities. As the student changes his watchstation console mode the holocube view could change to reflect the performance of that mode. This visual augmentation can assist teaching tactical deployment of offensive and defensive capabilities to incoming sailors. A multimedia hypertext tutorial is integrated into the virtual world detailing information about console operations, responsibilities, and workflow of watchstations in the CIC. The Virtual CIC application can be run in a variety of display and input configurations, including a simple single monitor with mouse and keyboard input for individual training, stereoscopic large multi-screen panoramic displays for group viewing or playback, and stereoscopic head mounted displays with hand and body movement tracking devices for fully immersive team training in order to meet different training needs.

Virtual consoles are easily reconfigured or rearranged as needed for different ship configurations for maximum efficiency. By combining the virtual environment with a ship simulation model, operational procedures under various tactical scenarios can be explored and refined easily. System performance can be stressed in a realistic scenario without the risking accidents or tying up the real operational hardware. Incorporated information visualization aids allowed students to learn and understand the operations much faster than in a regular classroom environment. Virtual reality based training also allows students to train with advanced systems before the hardware has been built, or try out new physical layouts that do not currently exist, as well as experiment with and develop new operational procedures. The system described can be applied to land-based or mobile command centers as well as the ship CIC. Also, it is not limited to training applications. Currently, we are extending the system for conducting virtual prototyping (i.e., design and develop new systems in a virtual environment before building the real physical system).

3. Virtual CIC with RTI

This section presents the software development process and the test procedure used in embedding RTI v.1.3r6 into Virtual CIC. Prior to any development,

the test team contacted and collaborated with personnel from the Joint Advanced Distributed Simulation (JADS) Joint Test & Evaluation (JT&S) Office and benefited from their experience [2]. JADS group has developed several HLA test software tools for their work including tools to log RTI activity that was applicable to this study also. The complete line of software tools can be downloaded from the JADS World Wide Web site [3]. Following sections present the experiences in adapting RTI version 1.3r6 as the network communication tool for the Virtual CIC.

3.1 Design the FOM and SOM

Network communication requirements were relatively clear from the existing implementation of the Virtual CIC. The Virtual CIC communication was based on passing data between CICs using generic strings, which were then parsed by each application and interpreted appropriately. This communication scheme could be very easily replaced by RTI interactions where the string would be passed as the parameter to an interaction. This, however, would make Virtual CIC federation a closed system. Another application would not be able decipher the meaning of those strings without access to Virtual CIC source code or specifications of the message strings.

Therefore, it was more desirable to design the Federation Object Model (FOM) and Simulation Object Model (SOM) with the idea that a foreign, yet non-existent, system would want to participate in the Virtual CIC environment, publish and subscribe data as appropriate. To that end, the events in the Virtual CIC are defined as appropriate RTI interactions, e.g. *touch*, *move*.

3.2 RTI-Virtual CIC Interface

Virtual CIC application is written using a well-structured, object-oriented VR Framework. A framework is a library of classes that embodies an object-oriented abstract design to solve a set of similar problems. Libraries typically provide algorithms, e.g. trigonometric functions in the math libraries. Frameworks, on the other hand, provide or sometimes enforce structure to an application in addition to algorithms. The VR Framework provides classes that handle communication between framework applications. These classes use events internally and hide the implementation of that protocol from the VR developer. Similarly, classes that use RTI as the communication protocol had to be added to the VR Framework. Then, Virtual CIC would require minimal changes to take of advantage of these new capabilities. System design was carried out with this vision.

The basic design of the system is such that there is a single external data distribution manager object for using the RTI services from the Virtual CIC. This is illustrated in Figure 3.1 with a Unified Modeling Language (UML) class diagram [4].

The blocks in UML diagrams represent classes with class names written in the top section of the block. When the name is printed in Italics, the class is said to be abstract. Abstract classes typically define interfaces. The lines connecting classes identify associations and the label on the line is the name of the association. The labels next to the class names, if they exist, identify the role of the class in the association.

ExtDataDistMgr is the only class Virtual CIC knows and deals with to communicate with the rest of the world whether that is another Virtual CIC program or any federate using the same Federation Object Model (FOM). Conversely, ExtDataDistMgr only knows of one instance of the CicCallback class—a class derived from CicCallback to be perfectly accurate since abstract classes can not be instantiated. The numbers "1" at both ends of the association indicate that the relation is one-to-one. A more detailed description of the interface design is shown in Fig ure 3.2, including the methods defined by the interface classes.

Figure 3.1: UML class diagram for the VCIC interface

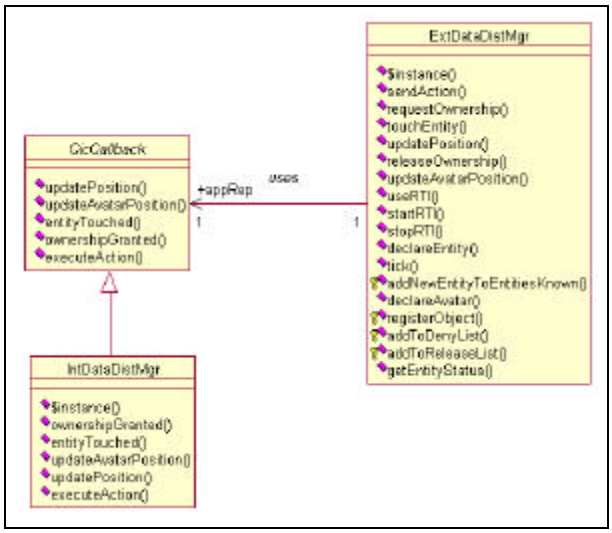

Figure 3.2: Class diagram for the Virtual CIC RTI module

3.3 Network Architecture

In order to isolate the computers from excess network traffic that may alter the results, they are placed in a separate, stand-alone network. The network architecture used in the testing is shown in Figure 3.3 along with the computers used in testing. The gray-shaded computers were not used in the testing as they were not capable of running the Virtual CIC application but they are shown in the diagram for completeness. The numbers printed above the connections in the diagram represent whether the connection is active or not. This notation is used to avoid duplicating the image several times. The numbers indicate when the connection would be live with respect to the number of federates participating in the federation. For example, when a federation with two federates is being tested, Seahawk and Falcon would be connected to the hub and there would be no other connection to the hub. Similarly, when three federates are being tested connections labeled "2+" and "3+" would be active, creating a standalone network of three computers connected through a 10BaseT hub. In the case of seven or more federates participating in the federation, the network becomes connected to the rest of the LAN and Internet but the effect of this proved to be negligible.

Clearly, performance results that we would measure during testing would depend on computer hardware characteristics, among other things. In other words, the faster the computer, the faster the RTI performance will be. This is because RTI is a software component that shares the resources of the host computer. In addition, the faster the computer can accomplish its simulation tasks, the faster or the more frequently it can give RTI a chance to perform the necessary tasks. Because of this consideration, it was desirable to use identical or very similar computers in the federation. Therefore, first four federates ran on identical hardware. Later, faster computers were also added to the federation. Virtual CIC is a virtual reality application that requires high-performance computers that provide high frame rate to give the user a realistic immersive environment. Therefore, any computer less powerful than an SGI Indigo2 was not used in the testing.

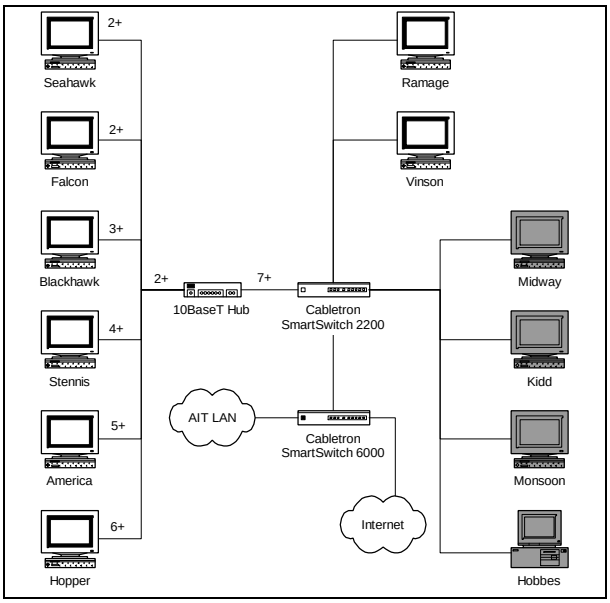

Figure 3.3: Test network configuration

3.4 RTI Interface Module

Virtual CIC simulations run in a distributed environment where multiple crewmembers (users) share the same virtual environment. It is important to maintain a consistent view of the virtual environment not to confuse the users. For example, if two crewmembers press the same button at the same time to turn the radar on, they may see that the radar did not come on at all. This is because there was two button push events generated, and the first turned the radar on, the second turned it off. To avoid this kind of behavior, resources can be allocated to particular users at particular times. Before pushing a button, the software (transparent to the user) requests ownership of that button from the federation. If the federation grants the ownership, the user would be allowed to push that button. Otherwise, it would get audio or

visual feedback that he or she is not allowed to push that button. This way, two crewmembers could not push the same button at the same time. Therefore, seamless exchange of ownership from federate to federate (or user to user) makes the Virtual CIC more user friendly. It was desirable to achieve the ownership exchange under 500 milliseconds to make the Virtual CIC experience more pleasant.

RTI provides three mechanisms to exchange ownership between federates [5]. These three mechanisms are shown as a UML sequence diagram in Figures 3.4–3.6. In these diagrams, RTI represent the RTI services offered by the Local RTI Component (LRC) and the Green or Yellow federate is any generic federate using these services.

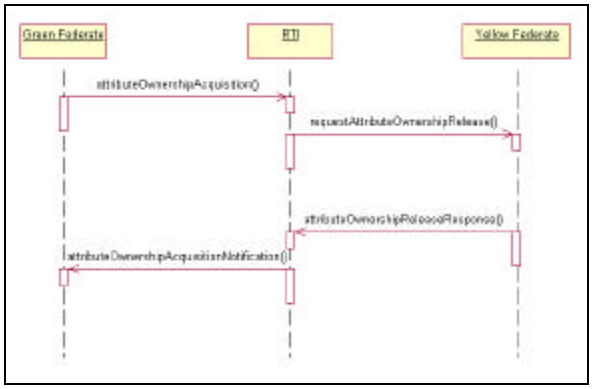

Figure 3.4: Ownership Pull (intrusive)

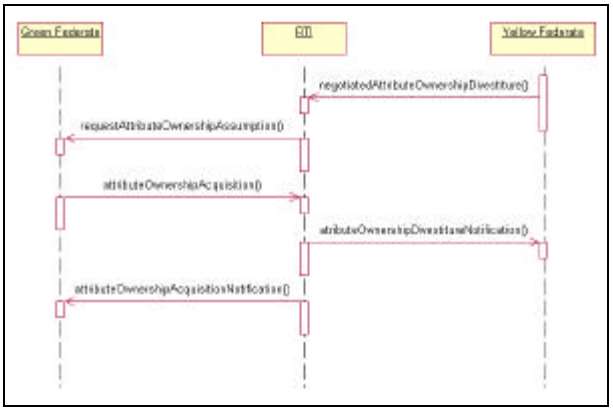

Figure 3.5. Ownership Push

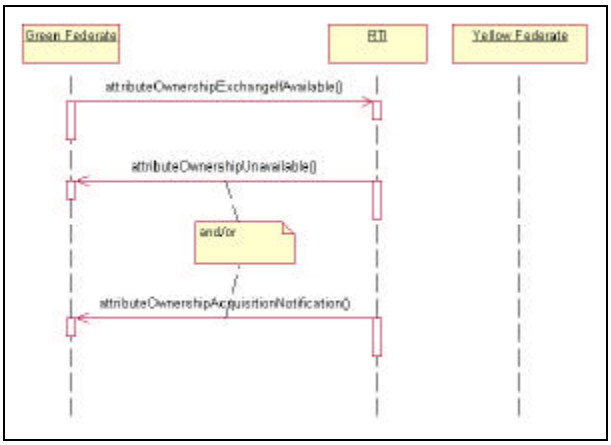

Figure 3.6: Ownership Pull (orphaned attributes)

The first scheme shown in Figure 3.4 involves green federate asking ownership of one or more attributes of an object from RTI. RTI, in turn, will call one of yellow federate's callback functions (assuming that the yellow federate owns those attributes in this example). Yellow federate then lets RTI know which of those attributes it is willing to give up. Then, R TI informs the green federate about for which attributes it has received the ownership. This scheme is very close to what was needed for Virtual CIC application. However, RTI implementation is limited in that if the yellow federate is not willing to give up ownership on any of the requested attributes, it has no way of informing the green federate of its decision. We tried using the *attributeOwnershipReleaseResponse* function with an empty list of attributes, which normally would contain the list of attri butes the yellow federate was giving up, but that call had no effect. In other words, it did not cause RTI to call the green federate's callback function *attributeOwnership-AcquisitionNotification* with an empty list of attributes. This is because RTI implementation "optimized out" the call, i.e. ignored it assuming that it has no consequence. This behavior was classified as normal or expected by the RTI Help Desk. The consequence of all of this is that green federate is never going to receive feedback about the request it made when yellow federate decides not to release any of the attributes. RTI developers asserted that this is in line with the HLA Interface Specification (I/F Spec) [6]. This assertion may be debated however as the I/F Spec is wide open for interpretation on this issue:

> "The *AttributeOwnershipReleaseResponse* service shall notify the RTI that the federate is willing to release ownership of the specified instance attributes for the specified object instance. The federate shall use this service to provide an answer to the question

> posed as a result of the RTI invocation of *Request-AttributeOwnershipRelease*. The returned argument shall indicate the instance attributes for which ownership was actually released. Completing of the invocation of this service shall be viewed as an implied *AttributeOwnershipDivestitureNotification* invocation for all of the instance attributes in the returned argument."
>
> —HLA Interface Specification

As underlined above (not underlined in the original document), the I/F Spec requires that the *attribute-OwnershipReleaseResponse* shall be used to respond to ownership requests. However, in the current implementation of the RTI this service can not be used to answer the request when the answer is "I do not want to give up the ownership of any of the requested attributes." Therefore according to this interpretation it can be argued that RTI does not perform as prescribed by the I/F Spec.

One way to get around this deficiency in the RTI behavior is to implement a time-out procedure. If a federate does not hear a response to its ownership request within a "reasonable" amount of time, it should assume that the request was denied. This time-out threshold depends on what kind of latency is expected in the network. It can be calculated by adding the maximum expected delay between two nodes and an additional duration to account for unusual behavior on the network. For Virtual CIC, this time-out value is set to 500 milliseconds, as this was our maximum response time requirement.

The other two ownership exchange schemes that were not needed in Virtual CIC but could be used in other applications are shown in Figures 3.5 and 3.6. Figure 3.5 shows how a federate can divest its ownership of one or more attributes. (There is also an unconditional divestiture method that is very similar to what's shown in the figure.) Figure 3.6 illustrates the procedure to retrieve ownership for attributes that are available. These attributes may be ones that were released by previous federates without properly transferring the ownership to another federate. In such cases, RTI maintains the ownership for the attributes.

One desirable but missing functionality identified in the RTI was the capability to request ownership for the object as a whole instead of a set of its attributes. Owning an object as a whole can be considered owning all its attributes. As such, requesting all the attributes of an object is the same as requesting the ownership for the object and RTI provides this functionality. This works well when only one federate requests ownership of all the attributes at any given time. However, if two or more federates request ownership of all the attributes of an object at about the same time, each federate may get a subset only. In other words, there is a chance none of the federates will own the object as a whole. This issue is also addressed by Kuijpers, *et al* [7]. In Virtual CIC application, this "fight for all attributes" was avoided by choosing one attribute to represent ownership of an object in the federation. This could be an attribute defined in the FOM for this purpose specifically. Instead, it was decided to use the privilegeToDelete attribute already defined for all attributes by the RTI.

3.5 Ownership Exchange Performance Test Results

Initially, the testing involved two identical SGI Indigo2 computers. The participants walked over to a console in the virtual world and pressed each button and saw the effect. If the other person owned the button at the time, nothing happened. Otherwise, the radar may have been turned on, or the scale on the radar screen may have been changed depending on what button was pushed. During this process, it was observed whether there was a consistent picture for both participants. In other words, if the radar appeared on for one user, it was on for the other also. Maintaining tactical picture consistency was one of the requirements and RTI fulfilled this requirement using *reliable* communication mode.

It was expected that as the number of federates increase, the ownership request response time would increase. To some extent, this prediction was true as shown in Figure 3.7. The figure shows a scatter plot of ownership exchange response time where the legend shows the name of the computer where the federate was running. Ownership exchange response time here is defined as the time from the point the user pushes a button and causes an ownership request to the time he or she gets a response from the federation. Considering the fact that RTI is single-threaded, in other words, does not function while program control is not explicitly given to it, the response time includes not only the RTI performance but also the Virtual CIC performance characteristics such as rendering speed. In fact, this is what makes measured response times depend on the computer platform.

Since RTI does not run as a separate thread in an application, it has to be given CPU cycles as frequently as the data must be exchanged. This is achieved by calling the RTI tick() function. Virtual CIC gave RTI

ample opportunity to perform its tasks by calling the tick function between rendering of each frame so that communication between the federates would not be hindered.

Figure 3.8 shows the average response time for ownership requests as the number of federates in the federation increase. The first four federates that participated on this testing procedure was running on four identical SGI Indogo2 computers with Maximum Impact graphics board and a 250 MHz IP22 processor (*Seahawk*, *Falcon*, *Blackhawk* and *Stennis* in Figure 3.7). The fifth federate ran on another SGI Indogo2 computer with a High Impact graphics board and a 195 MHz IP28 processor (*America* in Figure 3.7). The next three additions to the federation were even faster computers, which explains the reduction in average response time. Table 3.1 shows a summary of the computer hardware used in testing.

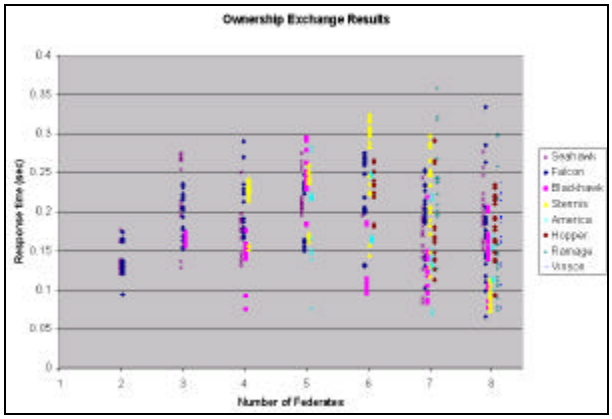

Figure 3.7: Ownership exchange response times

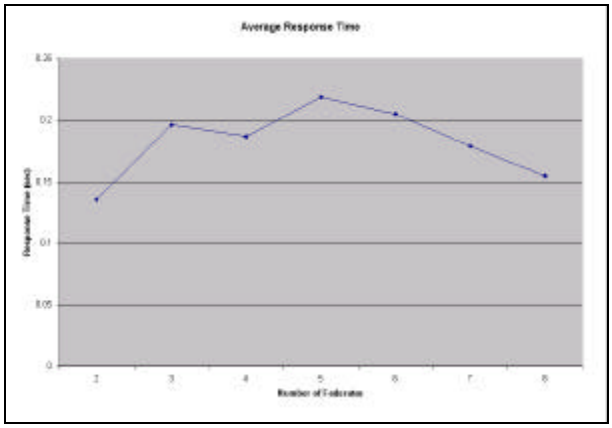

Figure 3.8: Average ownership exchange response time

Table 3.1: SGI hardware that was used to run Virtual CIC

		Number and type of Processor (CPU)			
Seahawk	Indigo2	1 x 250MHz MIPS R4400	Maximim Impact	256	Ethernet
Falcon	Indigo2	1 x 250MHz MIPS R4400	Maximum Impact	256	Ethernet
Blackhawk	Indigo2	1 x 250MHz MIPS R4400	Maximum Impact	256	Ethernet
Stennis	Indigo2	1 x 250MHz MIPS R4400	Maximum Impact	256	Ethernet
America	Indigo2	1 x 195MHz MIPS R10000	High Impact	256	Ethernet
Hopper	Octane	2 x 195MHz MIPS R10000	MXI	128	Fast Ethernet
Ramage	Octane	1 x 195MHz MIPS R10000	MXI	256	Fast Ethernet
Vinson	Onyx	4 x 194MHz MIPS R10000	Infinite-Reality	512	Ethernet

The performance on a WAN is also measured. Assuming that two federates are spread across the country, the additional time required to transfer data back and forth is approximately 60 milliseconds. (For example, roundtrip transfer of 50-to-1000 bytes of data between Washington, D.C. and Mountain View, CA takes 60-to-65 milliseconds.) To achieve the effect of a cross-country WAN between two federates, they are connected through two routers and a 30-millisecond delay is introduced to each packet during the transmission between the routers. With this configuration, the average response time was measured as 270 milliseconds. This is acceptable as it is well below our 500-millisecond acceptable limit. The only concern though, is that the RTI's failure to provide negative feedback on ownership requests. When running a federation on a WAN, the time-out technique described earlier may be hazardous as some requests can take too long to be responded as the network temporarily becomes unavailable, which is certainly more likely on a WAN than a LAN. Therefore time-out period would have to be kept long. This period may have to be too long to account for low quality connections, and may deem running of the federation on a WAN impractical.

4. Discussions

The performance levels obtained using RTI for network communication instead of UDP in Virtual CIC application was acceptable. Most of the CPU time spent from the time ownership request was made until the request response was received was spent by the Virtual CIC application. It was measured that the time RTI takes to respond to ownership requests was about 15 milliseconds, compared to an average of 200

millisecond response time for the user to know whether he or she gets the ownership to an entity. Therefore, RTI would perform well within acceptable levels for this task even for more federates than tested. A response time of 500 milliseconds is considered acceptable for most Virtual CIC users.

The experience in developing an HLA compliant application using the RTI libraries developed by DMSO was relatively painless. The Application Program Interface (API) that was provided by the RTI library was sufficient in general. However we did experience difficulty in a couple of areas.

Firstly, there are three schemes provided by RTI to exchange ownership. According to one of the three schemes, a federate can request ownership for a set of attributes. The federate owning those attributes gets a callback from RTI to release them. When that federate responds to RTI by listing all the attributes it is willing to release, RTI lets the requesting federate what it is allowed to own. The problem is that even when the federate that owned the attributes know for sure that it is not willing to give up the ownership to any of these attributes, there is no way it can respond to the request as such. The consequence of this is that the requesting federate never hears anything from the RTI whether it will get the ownership for those attributes or not. This can easily be fixed by letting the owning federate to respond to RTI by giving an empty list of attributes as the attributes the requesting federate can own.

Second difficulty experienced in using RTI was that the part of the RTI code needed for interfacing with Virtual CIC appeared to have interfered with other software libraries Virtual CIC was using. Even when none of the RTI services were being used, simply linking the RTI libraries into a very simple IRIS Performer (a commercial VR development tool from SGI) [8] application created system crashes in a reproducible manner when a system call is made from the C++ application, e.g. `system("sfplay bell.aiff")`. This is certainly an unacceptable behavior. Without the source code for the local RTI component that the applications link with, it is virtually impossible to identify the source of the problem. However, through the use of a debugger, it appears that IRIS Performer and RTI are using conflicting signals that causes the system to crash. One possible reason for this to occur is that maybe RTI is using global data that gets initialized and in the process uses signals. It is not yet clear what the cause of such flaky behavior is.

There is also a need for better documentation of RTI software. For example, the "RTI.rid" file that comes with the RTI distribution contains a slew of parameters that can be modified to achieve different behavior, and possibly better or worse performance. These parameters are not explained in any documentation.

RTI distribution includes an electronic version of the programmer's manual in Adobe Acrobat ® format [9]. This is very useful to programmers. It would be even more useful however if the document included a hyperlinked index, so that navigation is faster and easier.

5. Acknowledgements

The authors thank the Navy Modeling and Simulation Office (OPNAV N6M) for sponsoring this study. In particular, we thank Mr. James Weatherly and Mr. George Phillips of N6M and Dr. Susan Numrich of NRL for their valuable input and review of the work. In addition, JADS staff has been helpful with their input early in the project.

6. References

[1] HLA World Wide Web site: http://hla.dmso.mil.

[2] D. L. Wright, C. J. Harris, J.W. Black: "High Level Architecture Runtime Infrastructure Test Report," JADS Joint Test & Evaluation, 11104 Menual N.E., Albuquerque, NM, August 1998.

[3] JADS World Wide Web site: http://www.jads.abq.com/html/jads/JADS.htm.

[4] Grady Booch, Ivar Jacobson and James Rumbaugh*; The Unified Modeling Language User Guide*, Addison Wesley, 1998.

[5] High Level Architecture Run-Time Infrastructure Programmer's Guide, RTI 1.3 Version 6, DMSO, March 12, 1999.

[6] High Level Architecture Interface Specification, Version 1.3, DMSO, April 2, 1998.

[7] N. Kuijpers, J. Lukkien, B. Huijbrechts, M. Brassé: "Applying Data Distribution Management and Ownership Management Services of the HLA Interface Specification," Proc. 1999 Fall Simulation Interoperability Workshop, 99F-SIW-023, Vol. 1, pp. 154–161, Orlando, FL, September 1999.

[8] *IRIS Performer Programmer's Guide*, Document Number 007-1680-040, SGI, Mountain View, CA, 1997.

[9] Adobe Acrobat, http://www.adobe.com/prodindex/acrobat/main.html.

Author Biographies

DR. SULEYMAN GULEYUPOGLU is a Research Scientist of ITT Industries, Inc., working at the U.S. Naval Research Laboratory as an on-site contractor. He received his Ph.D. degree in Engineering Science and Mechanics from The University of Alabama, Tuscaloosa, AL. His current research interests include distributed simulations, virtual reality, collaborative engineering, and expert systems.

PATRICK MELODY is a software developer of ITT Industries, Inc., working at the U.S. Naval Research Laboratory as an on-site contractor. He received his M.S. degree in Computer Science from North Carolina State University. Current interests include virtual reality and reliable software construction.

HENRY NG is the Head of the Visualization and Computing Systems Section of the Advanced Information Technology branch in Naval Research Laboratory. He has been actively involved in simulation and modeling over twenty years. Prior to joining NRL, he was the Head of the Simulation and Modeling branch of the Warfare Analysis department of NSWCDD in White Oak, Maryland. He was the principle architect of a large scale sea, space, and land battle force level simulation model known as MARS (Multi-warfare Assessment and Research System). In addition, Henry was a member of the Technical Support Team to support DMSO Architecture Management Group (AMG) to develop High Level Architecture during 1994 -1996